Programación II
Taller 2025

"Sistema de Gestión de F1: Escuderías Unidas"

- **Curso:** Programación 2
- **Profesores:**
  - Silva, Elizabeth
  - Aguirre, Juan José
  - Balbuena, Patricia
- **Autores:**
  - Trapote, Diego Alejandro
  - Toribio, Juan Francisco

---

1. Idea General del Sistema

El proyecto "Escuderías Unidas" es un sistema de escritorio (construido en Java Swing) diseñado para modernizar y centralizar la administración de competencias de Fórmula 1. La propuesta busca reemplazar la gestión manual de datos por una aplicación robusta que maneje todas las entidades clave del deporte.

El sistema administra:

- **Entidades Principales:** Pilotos, Escuderías, Autos, Mecánicos, Circuitos y Países.
- **Gestión de Competencias:** Permite planificar Carreras, inscribir pilotos a ellas (asignándoles un auto) y registrar los resultados finales (posiciones, vueltas rápidas).
- **Informes y Estadísticas:** Genera reportes clave, como el ranking de pilotos por puntos, el historial de podios, y análisis detallados por escudería o circuito.

A diferencia de un sistema web, este proyecto utiliza un modelo de **persistencia en memoria**. Todos los datos se almacenan en `ArrayLists` dentro de una clase de persistencia (`GestionDeDatos`), que se inicializa con un conjunto de datos de prueba (`cargarDatosDePrueba`) cada vez que se inicia la aplicación.

2. Objetivos del Proyecto

- **Objetivo General:** Desarrollar una aplicación de escritorio funcional en Java Swing que cumpla con todos los requisitos de registro, gestión e informes propuestos en el diagrama UML.
- **Objetivos Específicos:**
  - Implementar una **arquitectura de 3 capas** (Presentación, Lógica y Persistencia) para separar responsabilidades.
  - Realizar las operaciones **CRUD** (Crear, Registrar, Actualizar, Eliminar) para todas las entidades principales.
  - Implementar las **reglas de negocio** clave (ej: sistema de puntuación, validaciones de inscripción).
  - Asegurar la **integridad de los datos** en memoria, implementando correctamente los métodos `equals()` y `hashCode()` para el manejo de objetos.
  - Generar los **informes** solicitados, realizando los cálculos y filtros necesarios en la capa de lógica.

3. Arquitectura del Sistema

El proyecto está construido sobre una arquitectura limpia de 3 capas, con inyección de dependencias para asegurar una única fuente de verdad para los datos.

1. **Capa de Presentación (Paquete `GUI`):**
    - Compuesta por todas las ventanas `JFrame`.
    - Es la única capa que el usuario ve.
    - Su única responsabilidad es mostrar datos y capturar la entrada del usuario.
    - **No contiene lógica de negocio.**
    - Todas las ventanas (excepto `Inicio`) reciben la instancia de `Servicios` por el constructor.
2. **Capa de Lógica (Paquete `Servicios`):**
    - Es el "cerebro" de la aplicación.
    - Actúa como el **único intermediario** entre la GUI y la Persistencia.
    - Contiene **toda la lógica de negocio** (ej: `rankingPilotos()`, `inscribirPilotoEnCarrera()`, `guardarResultado()`).
    - Implementa todas las validaciones (ej: "El piloto ya está inscripto", "El auto ya está asignado").
    - Recibe la instancia de `GestionDeDatos` en su constructor.
3. **Capa de Persistencia (Paquete `Persistencia`):**
    - Representada por la clase `GestionDeDatos`.
    - Es la única clase que "toca" los `ArrayLists` de datos.
    - Carga los datos de prueba (`cargarDatosDePrueba()`).
    - Provee métodos básicos para acceder a los datos (ej: `getPilotos()`, `addAuto()`, `removeCarrera()`).

> **Flujo de Arquitectura:** Para evitar la creación de "objetos infinitos", la clase `Inicio` crea la **única** instancia de `GestionDeDatos` y la **única** instancia de `Servicios`. Esta instancia de `Servicios` se pasa (inyecta) a todas las ventanas que se abren, garantizando que toda la aplicación opere sobre la misma base de datos en memoria.

4. Interfaces de Usuario Creadas

Se ha desarrollado un conjunto completo de interfaces gráficas para cubrir todos los casos de uso:

Módulo Principal

- **`Inicio.java`:** Pantalla de bienvenida y menú principal que navega a "Gestión" o "Informes".
- **`Gestion.java`:** Menú secundario para acceder a las 7 pantallas de gestión (ver imagen).

Módulo de Gestión (CRUD)

- **`Gestion_Piloto.java`:** Pantalla principal para ver, buscar, modificar y eliminar pilotos.
- **`Registro_Pilotos.java`:** Formulario para crear un nuevo piloto.
- **`Modificar_Piloto.java`:** Formulario para editar un piloto existente (cargando sus datos).
- **`Gestion_Carreras.java`:** Pantalla principal para gestionar carreras, con filtro por rango de fechas (ver imagen).
- **`Registro_Carrera.java`:** Formulario para crear una nueva carrera (con `JDateChooser` y `JSpinner` para fecha/hora).
- **`Modificar_Carrera.java`:** Formulario para editar una carrera existente.
- **`Gestion_Autos.java`:** Pantalla principal para gestionar autos.
- **`Registro_Auto.java`:** Formulario para crear un nuevo auto y asignarlo a una escudería.
- **`Gestion_Escuderias.java`**, **`Gestion_Circuitos.java`**, **`Gestion_Paises.java`**, **`Gestion_Mecanicos.java`** (y sus respectivas ventanas de Registro y Modificación).

Módulo de Lógica de Negocio

- **`Inscripcion_Carrera.java`:** (ver imagen) Interfaz clave que permite inscribir pilotos a una carrera específica, validando reglas de negocio (pilotos duplicados, autos duplicados,

fecha de inscripción). También permite "Dar de Baja" a un piloto de la carrera.
- **`Registro_Resultados_Carreras.java`:** Interfaz para cargar los resultados finales (Posición y Vuelta Rápida) de una carrera ya disputada.

Módulo de Informes

- **`Informes.java`:** Menú con botones para lanzar cada informe.
- **`ResultadosCarreras.java`:** Muestra un informe detallado de resultados de carreras en un rango de fechas.
- **`Ranking_pilotos.java`:** Muestra la tabla de posiciones del campeonato, calculando los puntos (1º=25) en base a los resultados.
- **`Historico_Pilotos.java`:** Muestra el informe de podios y victorias por piloto, con filtro.
- **`Informe_Autos.java`:** Muestra qué autos usó una escudería y en qué carreras.
- **`Informe_Mecanico.java`:** Muestra los mecánicos de una escudería (con sus años de experiencia y especialidad).
- **`MetodoCantidadCarrerasEnUnCircuito.java`:** Informe que cuenta cuántas carreras se corrieron en un circuito.
- **`MetodoPilotosyCircuitos.java`:** Informe que cuenta cuántas veces un piloto específico corrió en un circuito específico.

5. Bitácora de Actividades (Resumen Lógico)

El desarrollo del proyecto siguió una secuencia lógica de 6 fases:

1. **Fase 1: Planificación y Modelo (UML)**
   - Análisis del diagrama UML.
   - Creación de todas las clases del paquete `Modelo` (ej: `Piloto.java`, `Carrera.java`, `Auto.java`, `Especialidad.enum`, etc.).
2. **Fase 2: Arquitectura y Persistencia**
   - Definición de la arquitectura de 3 capas.
   - Creación de las clases `Servicios` y `GestionDeDatos`.
   - Implementación de la persistencia en memoria (`ArrayLists`) en `GestionDeDatos`.
   - Creación del constructor `Servicios(GestionDeDatos gestion)` para implementar la Inyección de Dependencias.
   - Corrección de la clase `Inicio.java` para que actúe como el "dueño" de las instancias de servicio y persistencia.
3. **Fase 3: Implementación del CRUD (Gestión)**
   - Desarrollo de todas las ventanas `Gestion_...` (ej: `Gestion_Piloto`, `Gestion_Autos`).
   - Implementación de los métodos `cargarTabla()`, `buscar()`, `eliminar...()`.
   - Desarrollo de todas las ventanas de `Registro_...` y `Modificar_...` (ej: `Registro_Piloto`, `Modificar_Auto`).
   - Implementación de la lógica para pasar los ID (`valor`) o DNI entre las ventanas de gestión y modificación.
   - Se implementaron los métodos `equals()` y `hashCode()` en los modelos (ej: `Auto`, `Escuderia`, `Carrera`) para permitir búsquedas y comparaciones correctas.
4. **Fase 4: Implementación de Lógica de Negocio (Carreras)**
   - Desarrollo de la ventana `Inscripcion_Carrera`.
   - Implementación de las validaciones críticas en `Servicios.inscribirPilotoEnCarrera()` (piloto duplicado, auto duplicado, fecha de inscripción).
   - Implementación de `darDeBajaPiloto()` (borrado lógico de la inscripción).
   - Desarrollo de la ventana `Registro_Resultados_Carreras`.
   - Implementación de `Servicios.guardarResultado()` para actualizar un `AutoPiloto` existente con su posición final y vuelta rápida.
5. **Fase 5: Implementación de Informes (Reportes)**
   - Desarrollo de todas las ventanas del paquete `Informes`.
   - Implementación de la lógica de filtrado por fechas (`buscarCarrerasPorFechas`).
   - Implementación del cálculo de estadísticas (`calcularEstadisticas` para

podios/victorias).
    - Implementación de la lógica de filtrado por escudería (`getResultadosPorEscuderia`, `getMecanicosPorEscuderia`).
    - Implementación del informe `rankingPilotos()`, con el reinicio de puntos, el bucle de suma y el ordenamiento de la lista.
6. **Fase 6: Pruebas, Refinamiento y Documentación**
    - Creación de un conjunto de datos de prueba (`cargarDatosDePrueba`) robusto y 100% interconectado para probar todos los informes.
    - Corrección de bugs (ej: el bug de "objetos infinitos", el bug de los IDs en `0`, el bug de `JComboBox`).
    - Mejora de la interfaz de usuario (uso de `JDateChooser`, `JSpinner`, `GridLayout`).
    - Generación de la documentación Javadoc del proyecto.

7. Conclusión

El desarrollo del Taller "Escuderías Unidas" ha sido completado con éxito, cumpliendo con todos los requisitos funcionales establecidos en la propuesta. El sistema es capaz de gestionar las altas, bajas y modificaciones de todas las entidades y de generar los informes solicitados.

El desafío técnico más significativo fue la correcta implementación de la **arquitectura de 3 capas** con una única fuente de datos en memoria. La solución del "bug de objetos infinitos", mediante la **inyección de dependencias** (pasando la instancia `Servicios` a todas las ventanas desde `Inicio`), fue un aprendizaje clave.

Además, el proyecto reforzó la importancia de una correcta implementación del modelo de objetos, especialmente los métodos `equals()` y `hashCode()`, que demostraron ser críticos para el filtrado y la gestión de datos en `ArrayLists` (como se vio en los informes de escudería y en el borrado de inscripciones).